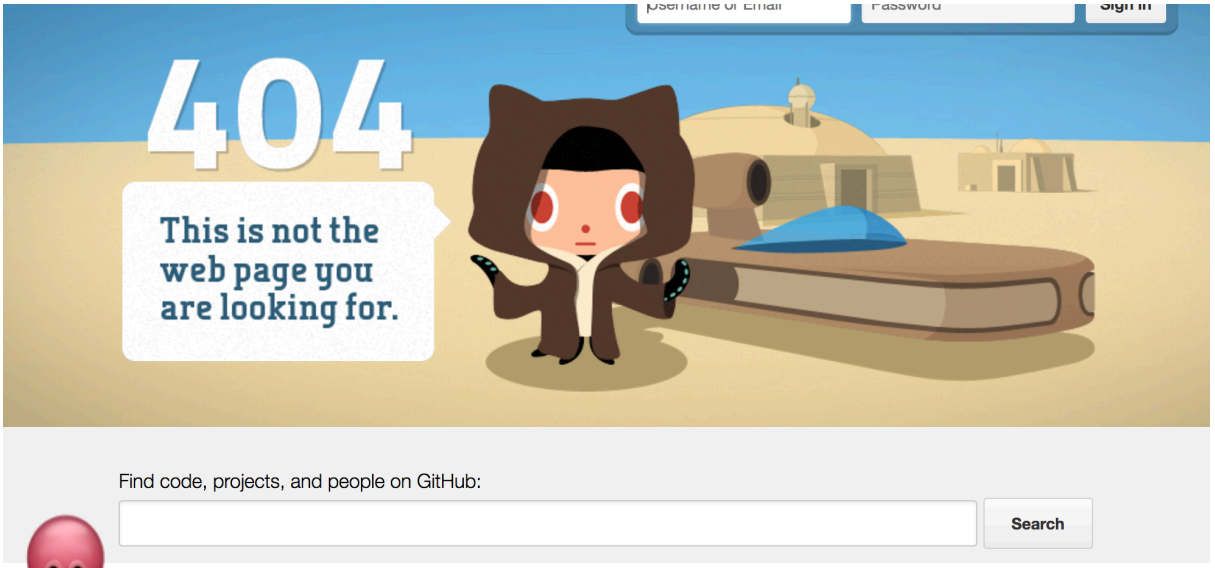


Информация по работе с GitHub

Tags Инструкция

Общая информация

GitHub — это веб-платформа, которая предоставляет услуги по хостингу и управлению кодом для разработки программного обеспечения. Она основана на системе контроля версий Git, которая позволяет разработчикам отслеживать изменения в коде, сотрудничать над проектами и управлять версиями программного обеспечения.

В нашей компании он используется для версионирования скриптов для отчетов, скриптов для выгрузки и загрузки БД.

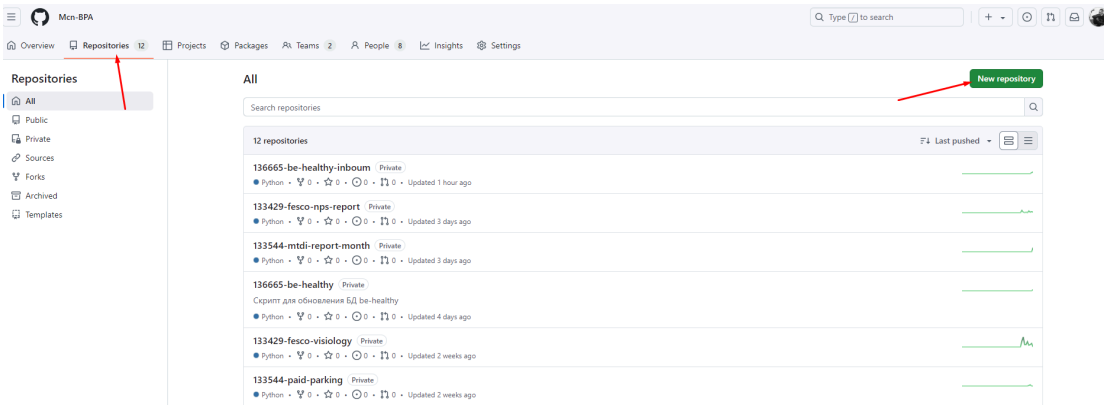
Все репозитории со скриптами находятся в организации по ссылке.

> !! Не редактируйте скрипты на сервере!!!
>
> Склонируйте репозиторий к себе и разрабатывайте новую версию там.

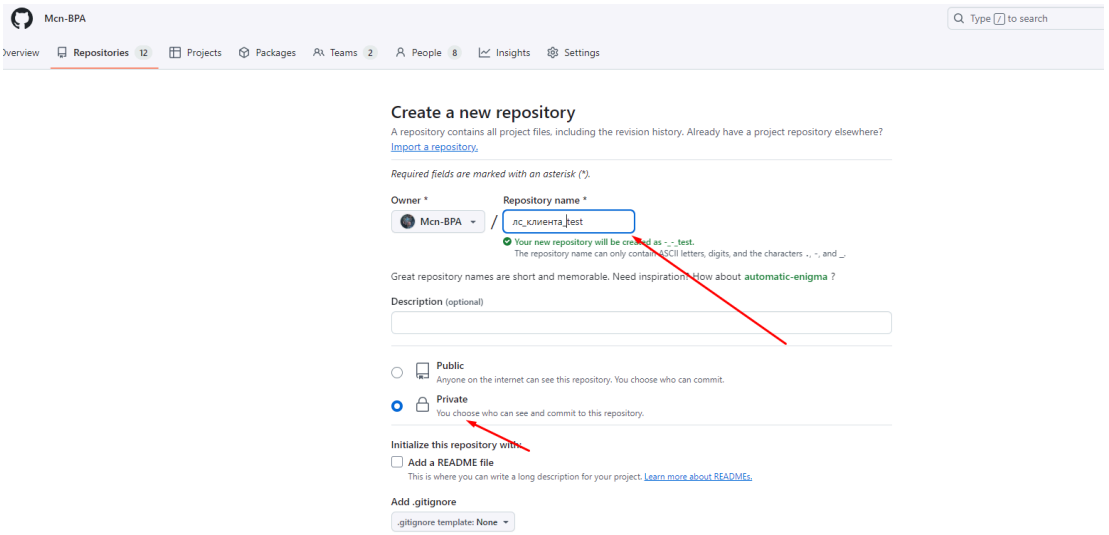
▼ Работа с GitHub в консоли

▼ Создание нового репозитория

Необходимо создать репозиторий в организации:



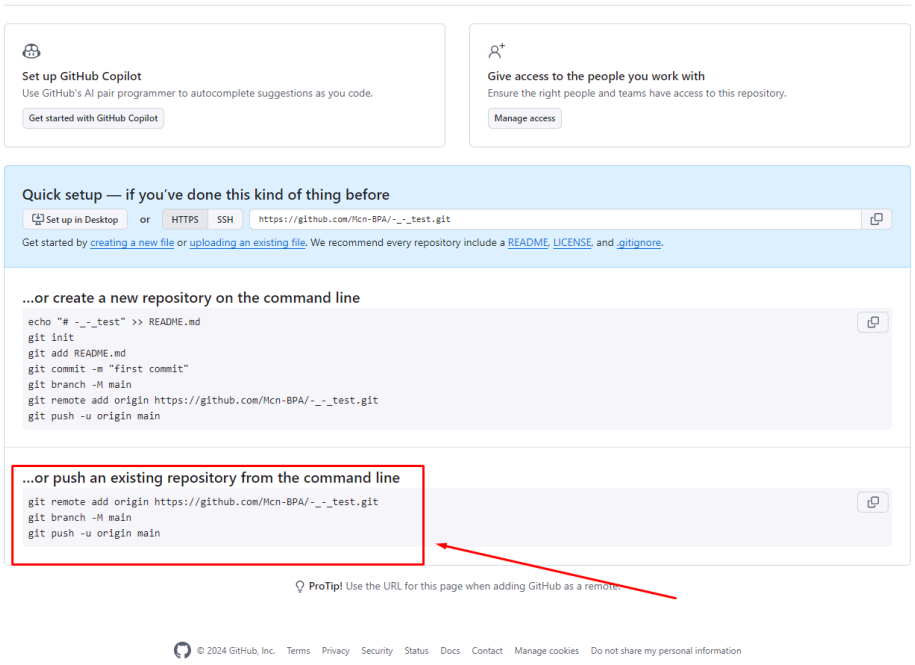
Заполняем названия, начиная его с лс клиента и проверяем, чтобы он был приватным:



После этого в консоли в папке проекта исполняем команду:

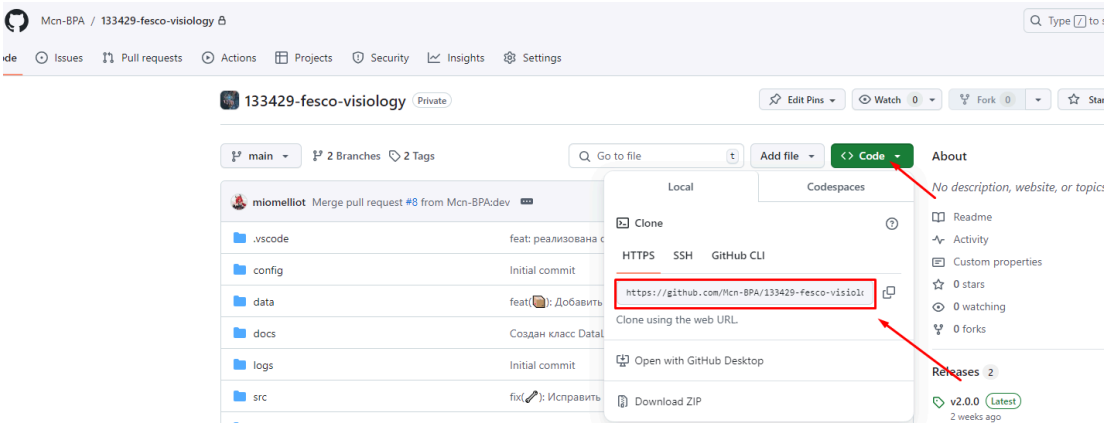
```
git init
```

После этого исполняем команды из github, которые высветились после



▼ Клонирование репозитория

Для того чтобы склонировать репозиторий необходимо знать его ссылку, для этого заходим в него:



Далее необходимо выполнить команду клонирования:

```
git clone ваша_ссылка_на_репозиторий
```

▼ Сохранение изменений

После того как все будет отредактировано и проверено нужно загрузить изменения на GitHub для этого нужно добавить измененные файлы:

```
git add .
```

Эта команда добавляет к отслеживанию все файлы.

Если нужно выбрать только один файл необходимо прописать следующее:

```
git add название_файла
```

После этого необходимо создать коммит (точку сохранения), для этого используется команда:

```
git commit -m "Ваш комментарий"
```

И после этого отправляем все это на сервер командой:

```
git push
```

▼ Изменение ссылки на удаленный репозиторий

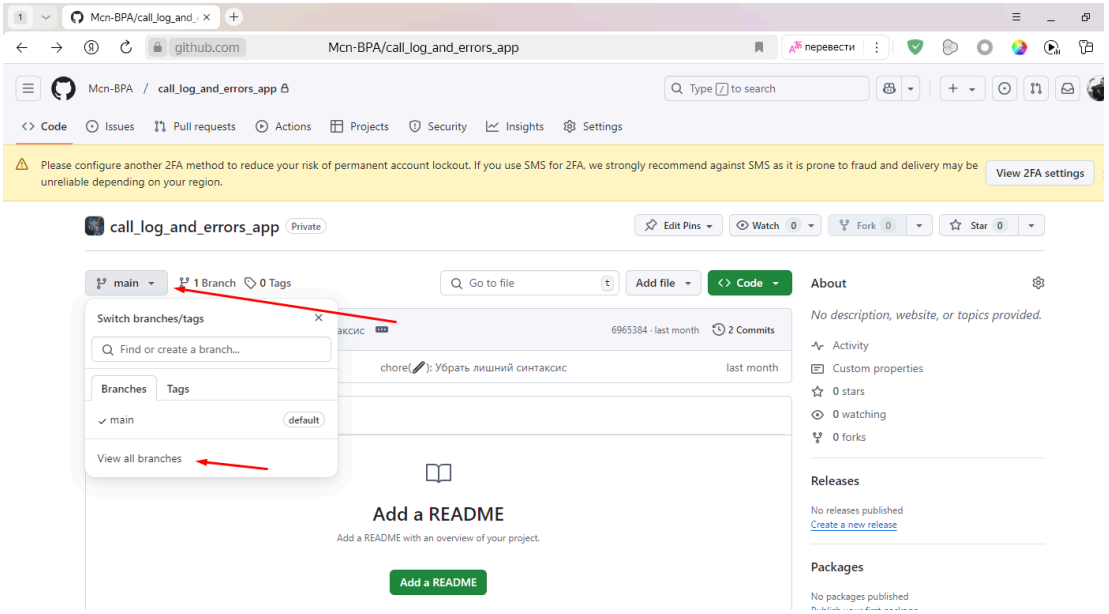
Иногда когда создаются скрипты вы могли склонировать репозиторий, но хотите сохранить его в новый репозиторий, для этого нужно либо удалить папку `.git`, но лучше изменить ссылку на репозиторий командой:

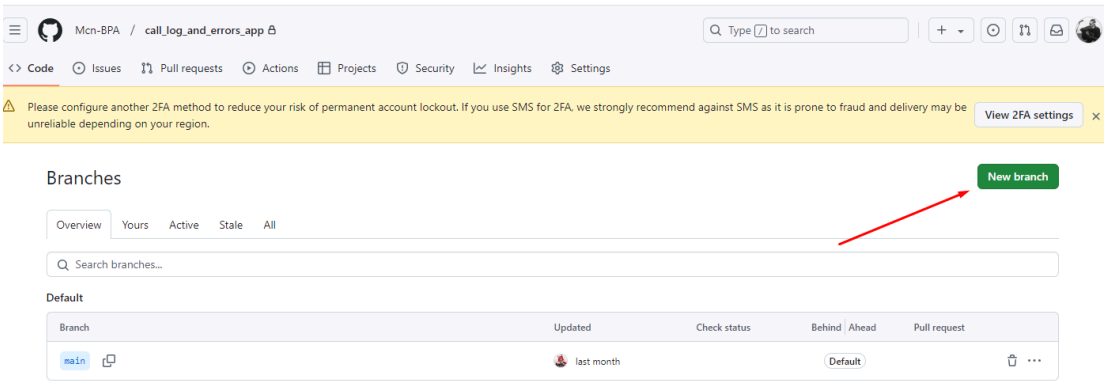
```
git remote set-url origin новая_ссылка
```

▼ 🐙Регламент по ведению версионности

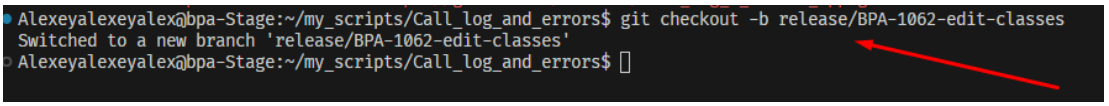
▼ Как делать pull request

Создайте новую ветку в проекте с соответствующим названием по инструкции.

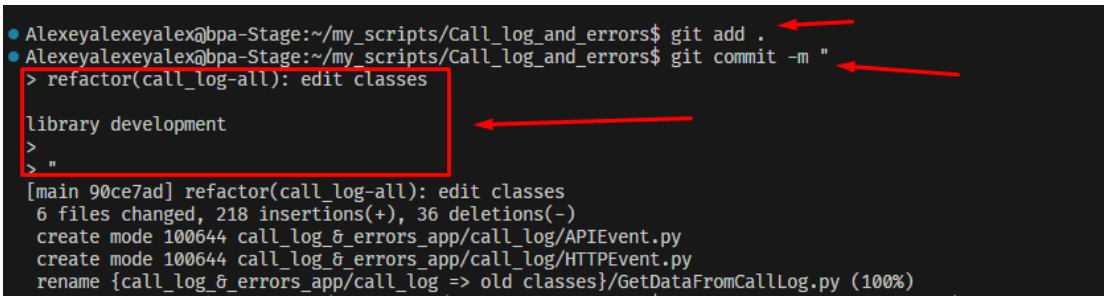




Создайте ветку в локальном проекте с аналогичным названием как и на сайте:



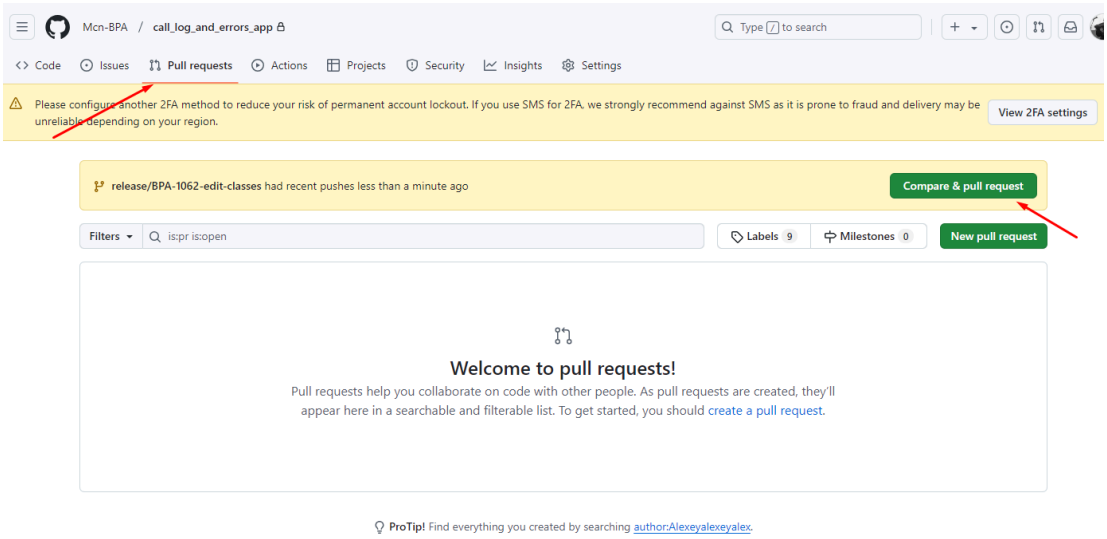
Добавьте измененные файлы и сделайте коммит согласно [правилам оформления]:



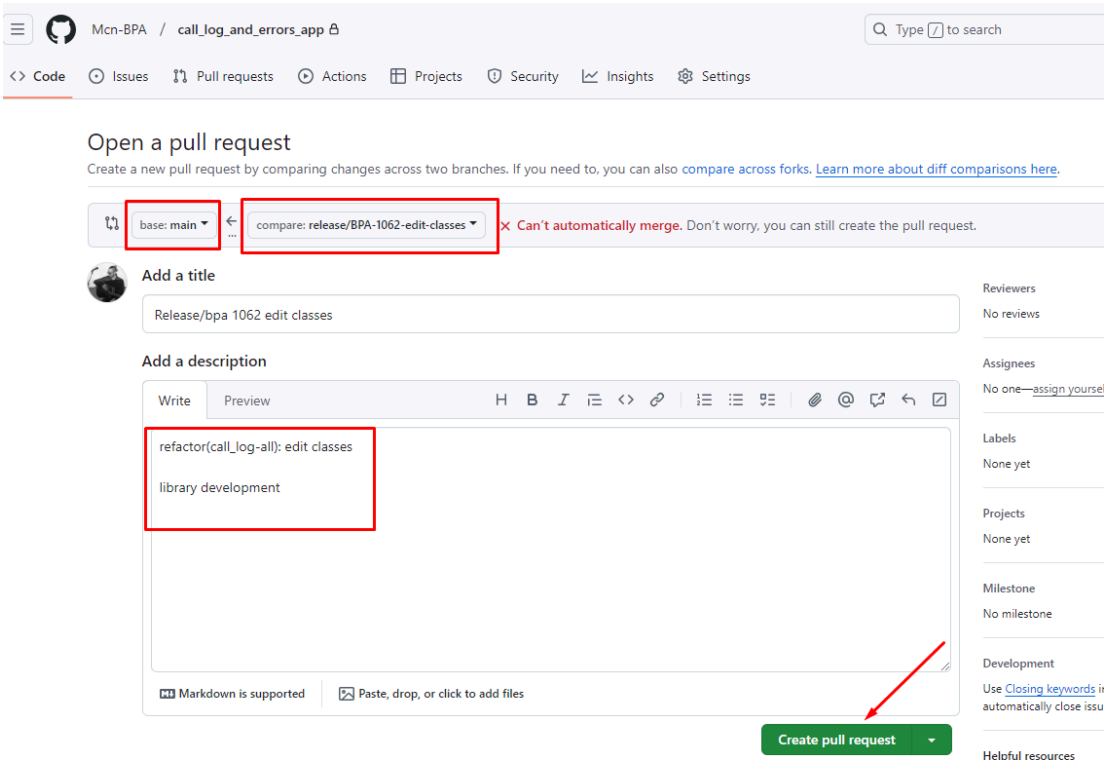
Дальше отправить изменения в удаленный репозиторий:

```
git push origin release/BPA-1062-edit-classes
git push origin ветка на удаленном репозитории
```

Далее на сайте появляется возможность создать pull request:



Далее проверьте ветки из какой в какую создается pull request, описать изменения и создать pull request:



▼ Правило именование новых веток проекта

Иногда назначение ветви сложно определить по одному лишь названию. Чтобы **явно** указать назначение ветви (исправление бага, новая функциональность, обновление документации и т.п.), принято использовать **префиксы**.

Наименование веток:

```
префикс/номер задачи jira-краткое описание
```

Пример:

```
docs/BPA-1062-update-readme
```

Вот общепринятые префиксы:

- **feature** – указывает на то, что в ветке разрабатывается новая фича, например **feature/add-filters** говорит о том, что в этой ветке идет работа по добавлению фильтров.
- **release** – используется для подготовки новых версий. Например, в ветке **release/v3.3.1-beta** делают последние правки и проверки перед выпуском новой версии: исправляют оставшиеся баги, обновляют документацию, меняют номер версии в коде и т.д. После того как подготовка завершена, содержимое ветки release сливают (**merge**) обратно в основную ветку. В результате в репозитории появляется новая версия кода со всеми изменениями из ветки release. Ее и выпускают как новый релиз.
- **bugfix** – применяется для исправления ошибок в коде. Обычно такая ветка создается при наличии конкретной проблемы (**issue**) с описанием найденного бага. Например, **bugfix/sign-in-flow** означает исправление ошибки в процессе входа в систему авторизации.
- **hotfix** – похож на bugfix, с одним важным отличием: **hotfix** используется для быстрого исправления критических багов, которые уже попали в продакшен. Например, **hotfix/cors-error** означает исправление срочной

ошибки CORS, связанной с междоменными запросами, которая сломала работу приложения в продакшене.

- **docs** – обозначает ветку, предназначенную для написания документации к проекту. Например, **docs/quick-start** – это ветка для написания инструкции по быстрому запуску проекта.

▼ **Шаблон коммита**

Для оформления сообщения коммита следует использовать следующий шаблон:

```
<type>(<scope>): <description>
<BLANK LINE>
<body>
<BLANK LINE>
<footer>
```

Type, scope и description вместе составляют заголовок коммита (header).

- **Type** - тип коммита (рефакторинг, исправление багов, новая фича и т.п.)
- **Scope** - область действия (где были изменения). Это может быть отдельный файл, директория или затронутая часть проекта (rendering, routing и т.д.). Не обязательно для заполнения (но желательно).
- **Description** - описание сути коммита. Обычно отвечает на вопрос "что было изменено/добавлено/удалено?" Например: add comment section
- **Blank line** - пустая строка, ей отделяется тело коммита от заголовка, и тело от футера.
- **Body** - не обязательно для заполнения. Здесь обычно отвечают на вопрос "Зачем были изменения?" и "Почему сделаны именно такие изменения?" (почему не стоит делать по-другому).
- **Footer** - не обязательно для заполнения. Может содержать информацию о критических изменениях (breaking changes), а также является местом для указания задач из бэклога, GitHub

issues, тикетов, и других проблем, которые этот коммит закрывает или с которыми он связан.

```
<тип>(<часть проекта>): <описание>
<пустая строка>
<Зачем были изменения?>
<пустая строка>
<критические изменения>
```

Тип коммита

- **feat** - используется при добавлении новой функциональности.
- **fix** - исправление багов.
- **refactor** - изменения кода, которые не исправляет баги и не добавляют функционал.
- **chore** - изменение конфигов, системы сборки, обновление зависимостей и т.д.
- **test** - всё, что связано с тестированием.
- **style** - исправление опечаток, изменение форматирования кода (переносы, отступы, точки с запятой и т.п.) без изменения смысла кода.
- **docs** - изменения только в документации.
- **perf** - изменения кода, повышающие производительность.
- **build** - изменения, влияющие на систему сборки или внешние зависимости (webpack, npm).
- **ci** - изменения в файлах конфигурации.

Пример коммита:

```
//header
chore: drop Node 6 from testing matrix

//body
see the issue for details on the typos fixed

//footer
```

```
BREAKING CHANGE: dropping Node 6 which hits end of life in April
closes issue #12
```

BREAKING CHANGE: указывается в футере и автоматически добавляется в конец заголовка. Критические изменения - это изменения, нарушающие обратную совместимость. Может быть частью коммита любого типа.

Должен начинаться с фразы `BREAKING CHANGE:`, за которой следует краткое изложение критического изменения, пустая строка и подробное описание критического изменения.

▼ Шаблон коммита 2.0

Новый подход к упрощённому и стандартизированному ведению версионирования Git с использованием эмодзи для упрощенного восприятия

▼ Принципы написания коммита

1. Используйте **Type** из шаблона
2. Уточняйте тип в разделе **Scope**
3. Пишите в повелительном наклонении в **Short description**

   пример: Создать файл `requirements.txt`
4. Если много изменений уточняйте в **Body**

   пример:

   - 📌 Создал ...

   - ✨ Обновил ...
5. Для особого взаимодействия используйте Footer
   a. `BREAKING CHANGE:` если изменение ломает обратную совместимость
   b. `Closes #123` для автоматического закрытия задачи в GitHub/GitLab
   c. `#45` для создания ссылки на задачу

для поиска смайликов в `win + ю` используйте `EU` раскладку клавиатуры

Эмодзи	Категория	Описание использования
🛠️ hammer and wrench	fix	Исправление бага, ошибки, некорректного поведения
✨ sparkles	feat	Добавление новой функциональности
🔧 wrench	refactor	Улучшение структуры кода без изменения логики
📄 page facing up	docs	Изменение или добавление документации
🗑️ wastebasket	**remove**	Удаление небольших частей кода, отдельных файлов, зависимостей, ненужных конфигураций
📦 package	chore	Обновление зависимостей, конфигурационных файлов
🚑 ambulance	hotfix	Критическое исправление, требующее немедленного деплоя
🏗️ building construction	WIP	Временный коммит, показывающий прогресс
🔥 fire	**remove**	Удаление большого объёма кода, глобальная чистка, удаление устаревшего или неиспользуемого кода, удаление временных решений (костылей)
🔍 mag	test	Обновление существующих тестов, рефакторинг, отладка
🚀 rocket	perf	Улучшение производительности
✅ white check mark	test	Написание новых тестов или исправление старых
⏳ hourglass flowing sand	deps	Обновление или замена зависимостей
🚧 construction	wip	Незавершённая работа, черновые изменения
🛑 stop_sign	security	Исправление уязвимостей и проблем безопасности
🎨 art	style	Изменения, не влияющие на код (отступы, форматирование)
🎭 performing arts	mock	Добавление или обновление мок-данных для тестирования

Эмодзи	Категория	Описание использования
💡 bulb	idea	Идеи и улучшения, предлагаемые в коде
🔄 arrows counterclockwise	revert	Откат коммита или изменений
🎯 dart	goal	Достижение поставленной цели в коде
⚙️ gear	ci	Изменения в конфигурации CI/CD (GitHub Actions, GitLab CI, Jenkins и т. д.)
🚦 vertical traffic light	ci	Исправление или настройка тестов в CI/CD пайплайне
🏭 factory	build	Обновления, влияющие на процесс сборки проекта
📜 scroll	release	Подготовка или выпуск новой версии
🚀 rocket	deploy	Связанное с деплоем изменений в продакшен или staging
🔄 counterclockwise arrows	workflow	Изменения в workflow автоматизации
🔐 locked with key	security	Улучшение безопасности пайплайна или контейнеров

▼ 🐙 Регламент завершения рабочего дня с GitHub

Каждый день перед выключением компа — чекни свой репозиторий.

- 🚀 **Есть прогресс?**

  Если ты сегодня красавчик и сделал что-то значимое (затащил фичу, прибил баг, обновил доку или зависимости) —

  сделай нормальный коммит с кратким описанием изменений.

  (Не надо пихать всё подряд в один коммит. Git умеет много, но не превращать твою кашу в порядок.)

- 🚧 **Не закончил? Оставил на завтра?**

  Зафиксируй всё в коммите 🚧 wip или 🏗️ WIP.

  Кратко опиши, что успел сделать. Пушь в GitHub. Не поленись.

Зачем это вообще надо ❓

- Чтобы утром не сидеть с пустым взглядом в код: "Чем я тут занимался?.."
- Чтобы не потерять результаты, если сервер внезапно решит сыграть в ящик.